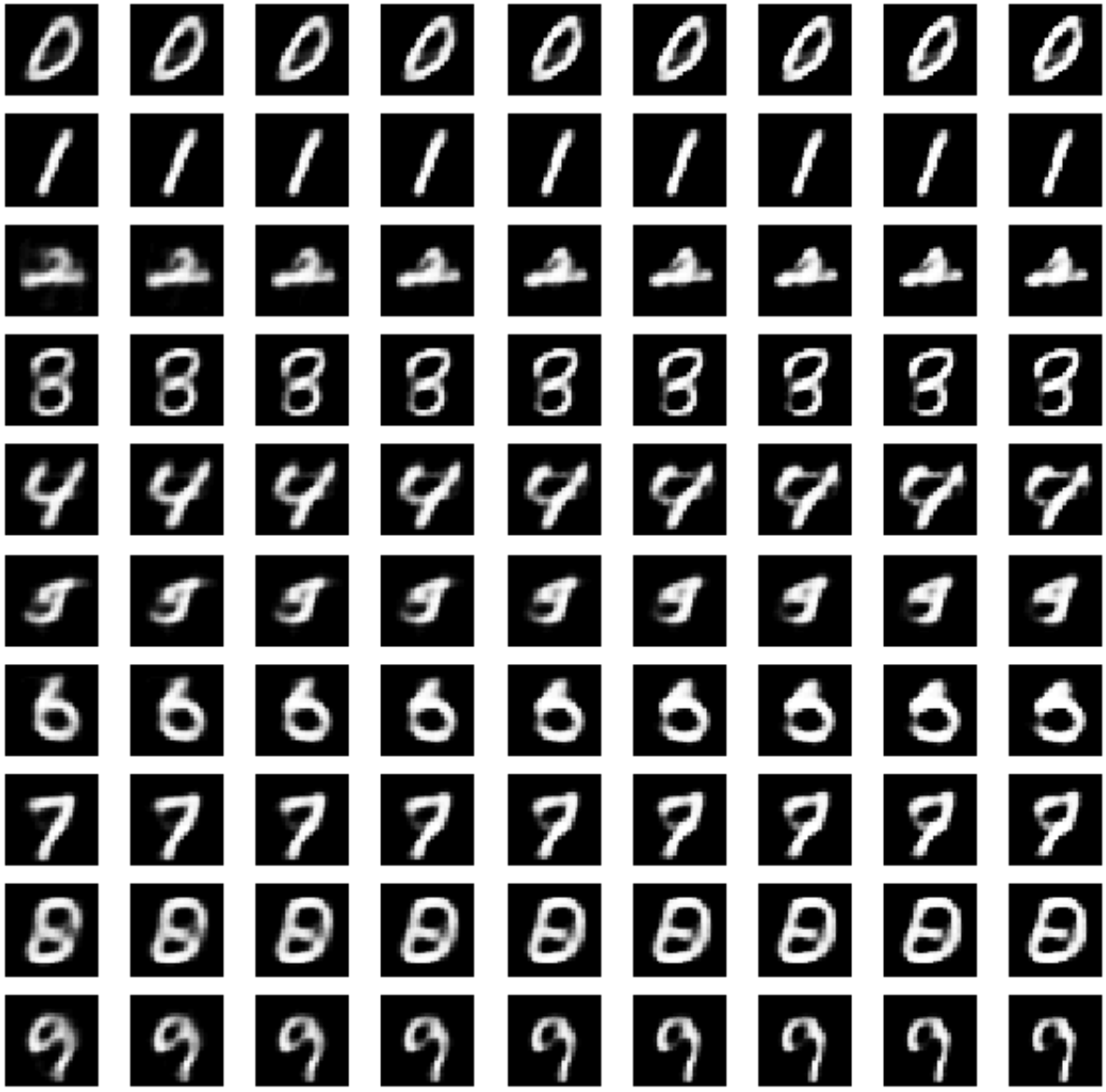
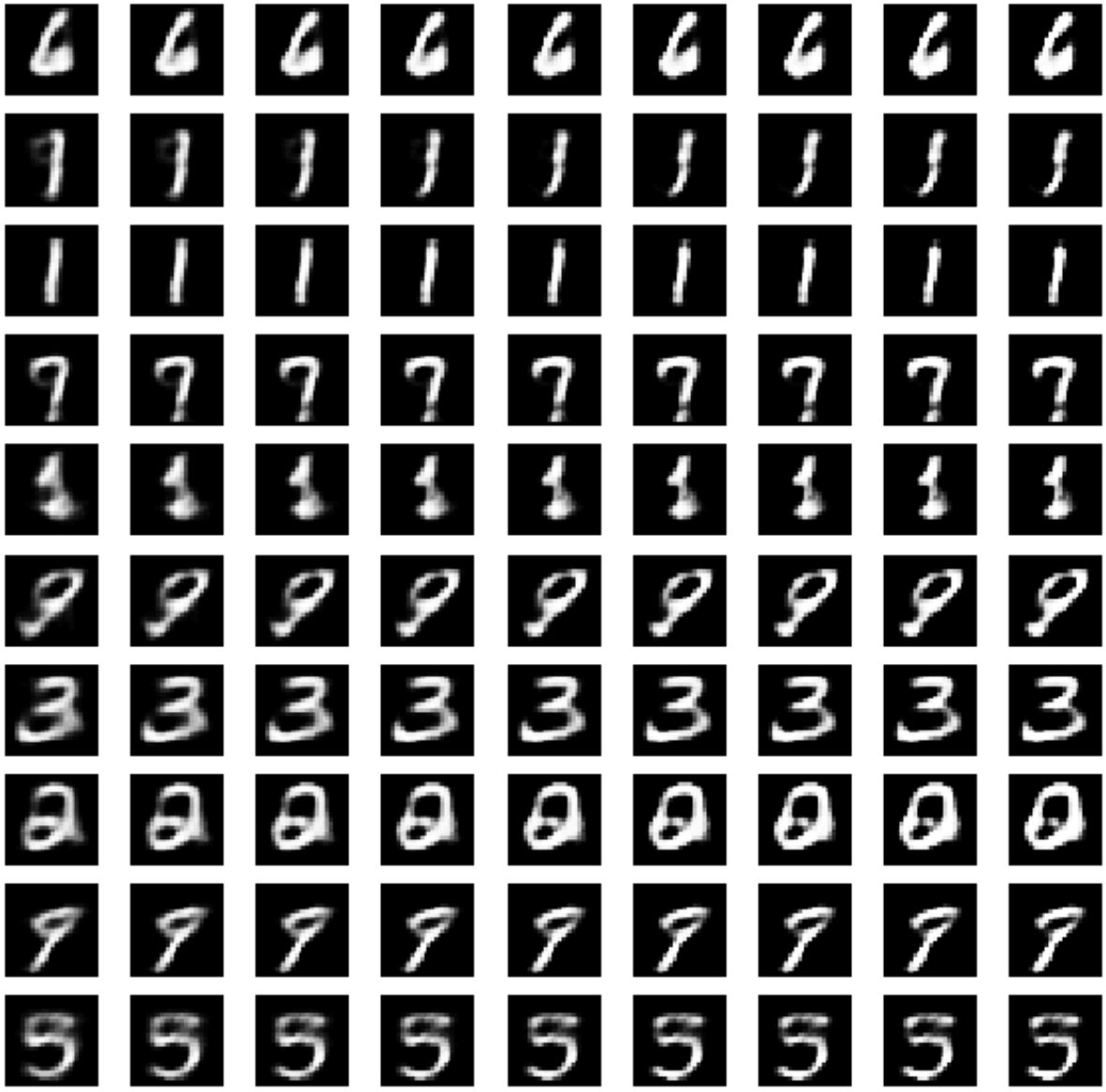
Page 1: Same digit interpolates



Page 2: Different digit interpolates



Page 3: Codes

```
from torchvision import datasets, transforms

## YOUR CODE HERE ##
transformations = transforms.Compose([transforms.ToTensor()])
mnist_train = datasets.MNIST(root = '/Users/xinqu/Sandbox/CS498 Applied Machine Learning/HW/HW9/hw9_data',
                             train = True, transform = transformations, download = True)
mnist_test = datasets.MNIST(root = '/Users/xinqu/Sandbox/CS498 Applied Machine Learning/HW/HW9/hw9_data',
                             train = False, transform = transformations, download = True)
```

```
from torch.utils.data import DataLoader

## YOUR CODE HERE ##
batch_size = 128
train_loader = DataLoader(dataset = mnist_train, shuffle = True, batch_size = batch_size)
test_loader = DataLoader(dataset = mnist_test, shuffle = True, batch_size = batch_size)
```

```
from torch.autograd import Variable
def to_var(x):
    return Variable(x)
def flatten(x):
    return to_var(x.view(x.size(0), -1))
```

```
from torch import nn
import torch.nn.functional as F
class VAE(nn.Module):
    def __init__(self, image_size=784, h_dim=400, z_dim=32):
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(image_size, h_dim),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim, z_dim*2)
        )

        self.decoder = nn.Sequential(
            nn.Linear(z_dim, h_dim),
            nn.ReLU(),
            nn.Linear(h_dim, image_size),
            nn.Sigmoid()
        )

    def reparameterize(self, mu, logvar):
        std = logvar.mul(0.5).exp_()
        esp = to_var(torch.randn(*mu.size()))
        z = mu + std * esp
        return z

    def forward(self, x):
        h = self.encoder(x)
        mu, logvar = torch.chunk(h, 2, dim=1)
        z = self.reparameterize(mu, logvar)
        return self.decoder(z), mu, logvar
vae = VAE()
vae
```

```
def vae_loss(x_hat, x, mu, logvar):
  ## YOUR CODE HERE ##
  # MSE LOSS + KL DIVERGENCE
    MSE = F.mse_loss(x_hat.view(-1, 784), x.view(-1, 784), reduction = 'sum')
    # - D_{KL} = 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return MSE + KLD
```

```
optimizer = torch.optim.Adam(vae.parameters(), lr=1e-3)
epochs = 10
for epoch in range(epochs):
    for idx, (images, _) in enumerate(train_loader):
        images = flatten(images)
        recon_images, mu, logvar = vae(images)
        loss = vae_loss(recon_images, images, mu, logvar)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("Epoch[{}/{}] Loss: {:.3f}".format(epoch+1, epochs, loss.item()/batch_size))
```

```
Epoch[1/10] Loss: 28.431
Epoch[2/10] Loss: 24.869
Epoch[3/10] Loss: 24.294
Epoch[4/10] Loss: 24.421
Epoch[5/10] Loss: 23.296
Epoch[6/10] Loss: 24.089
Epoch[7/10] Loss: 24.137
Epoch[8/10] Loss: 24.325
Epoch[9/10] Loss: 23.376
Epoch[10/10] Loss: 23.122
```

```
import matplotlib.pyplot as plt
from torchvision import utils
%matplotlib inline
import numpy as np

def create_interpolates(A, B, model):
  ## YOUR CODE HERE ##
    with torch.no_grad():
        imgs = []
        x, y, z = vae(flatten(A))
        mu = model.reparameterize(y, z)
        x1, y1, z1 = vae(flatten(B))
        mu1 = model.reparameterize(y1, z1)
        diff = mu - mu1 ###shape 1 * 32
        ###print(diff.shape)
        for i in range(9):
            tmp = mu + i / 8.0 * diff
            imgs.append(model.decoder(tmp))
        return imgs
```

```
similar_pairs = {}
for _, (x, y) in enumerate(test_loader):
    for i in range(len(y)):
        if y[i].item() not in similar_pairs:
            similar_pairs[y[i].item()] = []
        if len(similar_pairs[y[i].item()])<2:
            similar_pairs[y[i].item()].append(x[i])
```

```
vae.eval()
f, ax = plt.subplots(nrows = 10, ncols = 9, figsize = (10, 10))
for digit in range(10):
    imgs = create_interpolates(similar_pairs[digit][0], similar_pairs[digit][1], vae)
    for i in range(9):
        ax[digit, i].axis('off')
        image = imgs[i].numpy().reshape(28,28)
        ax[digit, i].imshow(image, cmap = 'gray')
```

```
random_pairs = {}
for _, (x, y) in enumerate(test_loader):
    # Make sure the batch size is greater than 20
    for i in range(10):
        random_pairs[i] = []
        random_pairs[i].append(x[2*i])
        random_pairs[i].append(x[2*i+1])
    break
```

```
f, ax = plt.subplots(nrows = 10, ncols = 9, figsize = (10, 10))
for digit in range(10):
    imgs = create_interpolates(random_pairs[digit][0], random_pairs[digit][1], vae)
    for i in range(9):
        ax[digit, i].axis('off')
        image = imgs[i].numpy().reshape(28,28)
        ax[digit, i].imshow(image, cmap = 'gray')
```